Microsoft Knowledge Base Article - 278973

SAMPLE: ExcelADO Demonstrates How to Use ADO to Read and Write Data in Excel Workbooks

Applies To

This article was previously published under Q278973

SUMMARY

The ExcelADO.exe sample illustrates how you can use ActiveX Data Objects (ADO) with the Microsoft Jet OLE DB 4.0 Provider to read and write data in Microsoft Excel workbooks.

MORE INFORMATION

The following file is available for download from the Microsoft Download Center:

ExcelADO.exe

Release Date: December 12, 2000

For additional information about how to download Microsoft Support files, click the following article number to view the article in the Microsoft Knowledge Base:

119591 How to Obtain Microsoft Support Files from Online Services

Microsoft scanned this file for viruses. Microsoft used the most current virus-detection software that was available on the date that the file was posted. The file is stored on security-enhanced servers that help to prevent any unauthorized changes to the file.

Why Use ADO?

The use of ADO to transfer data to or retrieve data from an Excel workbook gives you, the developer, several advantages over Automation to Excel:

- **Performance**. Microsoft Excel is an out-of-process ActiveX server. ADO runs in-process, and saves the overhead of costly out-of-process calls.
- **Scalability**. For Web applications, it is not always desirable to automate Microsoft Excel. ADO presents you with a more scaleable solution to handle data in a workbook.

ADO can be used strictly to transfer raw data to a workbook. You cannot use ADO to apply formats or formulas to cells. However, you can transfer data to a workbook that is pre-formatted and the format is maintained. If you require "conditional" formatting after the data is inserted, you can accomplish this formatting with Automation or with a macro in the workbook.

Jet OLE DB Provider Specifics for Excel Workbooks

The Microsoft Jet database engine can be used to access data in other database file formats, such as Excel workbooks, through installable Indexed Sequential Access Method (ISAM) drivers. In order to open external formats supported by the Microsoft Jet 4.0 OLE DB Provider, you specify the database type in the extended properties for the connection. The Jet OLE DB Provider supports the following database types for Microsoft Excel workbooks:

- Excel 3.0
- Excel 4.0
- Excel 5.0
- Excel 8.0

NOTE: Use the Excel 5.0 source database type for Microsoft Excel 5.0 and 7.0 (95) workbooks and use the Excel 8.0 source database type for Microsoft Excel 8.0 (97) and 9.0 (2000) workbooks. The ExcelADO.exe sample uses Excel workbooks in the Excel 97 and Excel 2000 format.

The following samples demonstrate an ADO connection to an Excel 97 (or 2000) workbook:

```
Dim oConn As New ADODB.Connection
With oConn
     .Provider = "Microsoft.Jet.OLEDB.4.0"
     .Properties("Extended Properties").Value = "Excel 8.0"
     .Open "C:\Book1.xls"
     '....
     .Close
End With
```

-or-

```
Dim oConn As New ADODB.Connection
oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
           "Data Source=C:\Book1.xls;" & _
           "Extended Properties=""Excel 8.0;"""
oConn.Close
```

Table Naming Conventions

There are several ways you can reference a table (or range) in an Excel workbook:

- Use the sheet name followed by a dollar sign (for example, [Sheet1$] or [My Worksheet$]). A workbook table that is referenced in this manner consists of the entire used range of the worksheet.

```
oRS.Open "Select * from [Sheet1$]", oConn, adOpenStatic
```

- Use a range with a defined name (for example, [Table1]).

```
oRS.Open "Select * from Table1", oConn, adOpenStatic
```

- Use a range with a specific address (for example, [Sheet1$A1:B10]).

```
oRS.Open "Select * from [Sheet1$A1:B10]", oConn, adOpenStatic
```

Table Headers

With Excel workbooks, the first row in a range is considered to be the header row (or field names) by default. If the first range does not contain headers, you can specify **HDR=NO** in the extended properties in your connection string. If the first row does not contain headers, the OLE DB provider automatically names the fields for you (where F1 would represent the first field, F2 would represent the second field, and so forth).

Data Types

Unlike a traditional database, there is no direct way to specify the data types for columns in Excel tables. Instead, the OLE DB provider scans a limited number of rows in a column to "guess" the data type for the field. The number of rows to scan defaults to eight (8) rows; you can change the number of rows to scan by specifying a value between one (1) and sixteen (16) for the **MAXSCANROWS** setting in the extended properties of your connection string.

Files Included with the Sample

The ExcelADO.exe file contains a Visual Basic Standard EXE project, Active Server Pages (ASP), Excel 97 and Excel 2000 Workbooks that act as templates, and a Microsoft Access 2000 database. The files included are as follows:

Visual Basic Standard EXE Project Files

- ExcelADO.vbp
- Form1.frm
- Form1.frx

Active Server Pages

- EmpData.asp
- Orders.asp

Microsoft Excel Workbooks

- OrdersTemplate.xls
- EmpDataTemplate.xls
- ProductsTemplate.xls
- SourceData.xls

Microsoft Access Database

- Data.mdb

How to Use the Sample

Extract the contents of the .exe file to a folder.

To use the Visual Basic project:

1. In Visual Basic, open the ExcelADO.vbp file.
2. On the **Project** menu, select **References**, and then set references to **Microsoft ADO Ext. for DDL and Security** and **Microsoft ActiveX Data Objects Library**. This sample code works with both ADO 2.5 and ADO 2.6, so select the version appropriate to your computer.
3. Press the F5 key to run the program. A form for the demonstration appears.
4. Click **Sample 1**. This sample creates a copy of OrdersTemplate.xls. It then uses ADO to connect to the workbook and opens a **Recordset** on a table that is a defined range in the workbook. The name of the range is **Orders_Table**. It uses ADO **AddNew**/**Update** methods to add records (or rows) to the defined range in the workbook. When the row additions are complete, the ADO **Connection** is closed and the workbook is displayed in Microsoft Excel. Follow these steps to do this:
    1. On the **Insert** menu in Excel, select **Names**, and then select **Define**.
    2. In the list of defined names, select **Orders_Table**. Note that the defined name has grown to include the newly added records. The defined name is used, in conjunction with Excel's OFFSET function, to compute a total on the data added to the worksheet.
    3. Quit Microsoft Excel and return to the Visual Basic application.
5. Click **Sample 2**. This sample creates a copy of EmpDataTemplate.xls. It uses ADO to connect to the workbook and uses the **Execute** method of the ADO connection to insert data (INSERT INTO in SQL) into the workbook. Data is added at defined ranges (or tables) in the workbook. When the data is transferred, the connection is closed and the workbook that results is displayed in Excel. After you examine the workbook, quit Microsoft Excel, and then return to the Visual Basic application.

6. Click **Sample 3**. This sample creates a copy of ProductsTemplate.xls. It uses Microsoft ADO Extensions 2.1 for DDL and Security object library (ADOX) to add a new table (or a new worksheet) to the workbook. An ADO **Recordset** is then obtained for the new table and data is added by using the **AddNew**/**Update** methods. When the row additions are complete, the ADO **Connection** is closed and the workbook is displayed in Excel. The workbook contains Visual Basic for Applications (VBA) macro code in the **Open** event for the Workbook. The macro runs when the workbook opens; if the new "Products" worksheet exists in the workbook, the macro code formats the worksheet and then the macro code is deleted. This technique presents a way for the Web developer to move formatting code away from the Web server and onto the client. A Web application could stream a formatted workbook that contains data to the client and allow macro code that would perform any "conditional" formatting that might not be possible in a template alone to run at the client.

    NOTE: To examine the macro code, view the **ThisWorkbook** module in the VBAProject for ProductsTemplate.xls.

7. Click **Sample 4**. This sample produces the same results as Sample 1, but the technique that is used to transfer the data is slightly different. In Sample 1, records (or rows) are added to the worksheet one at a time. Sample 4 adds the records in bulk by attaching the Excel table to an Access database and running an append query (or INSERT INTO..SELECT FROM) to append records from a table in the Access table to the Excel table. Once the transfer is complete, the Excel table is detached from the Access database and the workbook that results is displayed in Excel. Quit Excel, and return to the Visual Basic application.

8. The last sample illustrates how you can read data from an Excel workbook. Select a **table** in the drop-down list, and then click **Sample 5**. The Immediate window displays the contents of the table that you selected. If you select an entire worksheet ("Sheet1$" or "Sheet2$") for the table, the Immediate window displays the contents of the used range for that worksheet. Note that the used range does not necessarily begin on row 1, column 1 of the worksheet. The used range starts at the upper left-most cell in the worksheet that contains data.

   If you select a specific range address or a defined range, the Immediate window displays the contents of only that range on the worksheet.

To use the Active Server Pages (ASP):

1. Create a new folder named **ExcelADO** in the home directory of your Web server. Note that the default path for the home directory is C:\InetPut\WWWRoot.
2. Copy the following files to the folder you created in the previous step:
   - EmpData.asp
   - Orders.asp
   - Data.mdb
   - EmpDataTemplate.xls
   - OrdersTemplate.xls

3. The ASP scripts in this sample create copies of the workbook templates with the **Copy** method of the **FileSystemObject**. For the **Copy** method to succeed, the client that is accessing the script must have Write access to the folder that contains the ASP.
4. Navigate to Orders.asp (that is, http://YourServer/ExcelADO/Orders.ASP), and note that the browser displays an Excel workbook similar to the one in **Sample 1** of the Visual Basic application.
5. Navigate to EmpData.asp (that is, http://YourServer/ExcelADO/EmpData.ASP), and note that the browser displays an Excel workbook similar to the one in **Sample 2** of the Visual Basic application.

REFERENCES

For more information, see the white paper "Automating Office 97 and Office 2000" available on the following Microsoft Web site at:

http://support.microsoft.com/support/officedev/automation.asp

For additional information, click the article numbers below to view the articles in the Microsoft Knowledge Base:

195951 HOWTO: Query and Update Excel Data Using ADO From ASP

194124 PRB: Excel Values Returned as NULL Using DAO OpenRecordset

193998 HOWTO: Read and Display Binary Data in ASP

247412 INFO: Methods for Transferring Data to Excel from Visual Basic

257819 HOWTO: Use ADO with Excel Data from Visual Basic or VBA

The information in this article applies to:

- Microsoft Excel 2000
- ActiveX Data Objects (ADO) 2.5
- ActiveX Data Objects (ADO) 2.6
- Microsoft Visual Basic Professional Edition for Windows 6.0
- Microsoft Visual Basic Enterprise Edition for Windows 6.0
- Microsoft Active Server Pages

Last Reviewed: 12/15/2003 (5.0)
Keywords: kbAutomation kbfile kbProgramming KB278973

Send Print Help